

Vicious Circle Principle and Formation of Sets in ASP Based Languages

Michael Gelfond and Yuanlin Zhang

Texas Tech University, Lubbock, TX, USA
{michael.gelfond, y.zhang}@ttu.edu

Abstract. The paper continues the investigation of Poincare and Russel's Vicious Circle Principle (VCP) in the context of the design of logic programming languages with sets. We expand previously introduced language *Alog* with aggregates by allowing infinite sets and several additional set related constructs useful for knowledge representation and teaching. In addition, we propose an alternative formalization of the original VCP and incorporate it into the semantics of new language, *Slog*⁺, which allows more liberal construction of sets and their use in programming rules. We show that, for programs without disjunction and infinite sets, the formal semantics of aggregates in *Slog*⁺ coincides with that of several other known languages. Their intuitive and formal semantics, however, are based on quite different ideas and seem to be more involved than that of *Slog*⁺.

1 Introduction

This paper is the continuation of work started in [15] with introduction of *Alog* – a version of Answer Set Prolog (ASP) with aggregates. The semantics of *Alog* combines the Rationality Principle of ASP [12] with the adaptation of the Vicious Circle Principle (VCP) introduced by Poincare and Russel [29,31] in their attempt to resolve paradoxes of set theory. In *Alog*, the latter is used to deal with formation of sets and their legitimate use in program rules. To understand the difficulty addressed by *Alog* consider the following programs:

Example 1. P_0 consisting of a rule:

$p(1) :- \text{card}\{X: p(X)\} \neq 1.$

P_1 consisting of rules:

$p(1) :- p(0).$
 $p(0) :- p(1).$
 $p(1) :- \text{card}\{X: p(X)\} \neq 1.$

P_2 consisting of rules:

$p(1) :- \text{card}\{X: p(X)\} \geq 0.$

Even for these seemingly simple programs, there are different opinions about their meaning. To the best of our knowledge all ASP based semantics, including that of [6,34,15] view P_0 as a bad specification. It is inconsistent, i.e., has no answer sets. Opinions differ, however, about the meaning of the other two programs. [6] views P_1 as a reasonable specification having one answer set – $\{p(0), p(1)\}$. According to [34,15] P_1 is inconsistent. According to most semantics P_2 has one answer set, $\{p(1)\}$. *Alog*, however, views it as inconsistent.

As in the naive set theory, the difficulty in interpretations seems to be caused by self-reference. In both P_1 and P_2 , the definition of $p(1)$ references the set described in terms of p . It is, of course, not entirely clear how this type of differences can be resolved. Sometimes, further analysis can find convincing arguments in favor of one of the proposals. Sometimes, the analysis discovers that different approaches really model different language or world phenomena and are, hence, all useful in different contexts. We believe that the difficulty can be greatly alleviated if the designers of the language provide its users with as clear intuitive meaning of the new constructs as possible. Accordingly, the *set name* construct $\{X : p(X)\}$ of *Alog* denotes *the set of all objects believed by the rational agent associated with the program to satisfy property p* . (This reading is in line with the epistemic view of ASP connectives shared by the authors.) The difficulties with self-reference in *Alog* are resolved by putting the following intuitive restriction on the formation of sets¹:

An expression $\{X : p(X)\}$ denotes a set S only if for every t rational belief in $p(t)$ can be established without a reference to S , or equivalently, the reasoner's belief in $p(t)$ can not depend on existence of a set denoted by $\{X : p(X)\}$.

We view this restriction as a possible interpretation of VCP and refer to it as *Strong VCP*. Let us illustrate the intuition behind *Alog* set constructs.

Example 2. Let us consider programs from Example 1. P_0 clearly has no answer set since \emptyset does not satisfy its rule and there is no justification for believing in $p(1)$. P_1 is also inconsistent. To see that notice that the first two rules of the program limit our possibilities to $A_1 = \emptyset$ and $A_2 = \{p(0), p(1)\}$. In the first case $\{X : p(X)\}$ denotes \emptyset . But this contradicts the last rule of the program. A_1 cannot be an answer set of P_1 . In A_2 , $\{X : p(X)\}$ denotes $S = \{0, 1\}$. But this violates our form of VCP since the reasoner's beliefs in both, $p(0)$ and $p(1)$, cannot be established without reference to S . A_2 is not an answer set either. Now consider program P_2 . There are two candidate answer sets²: $A_1 = \emptyset$ and

¹ It is again similar to set theory where the difficulty is normally avoided by restricting comprehension axioms guaranteeing existence of sets denoted by expressions of the form $\{X : p(X)\}$. In ASP such restrictions are encoded in the definition of answer sets.

² By a candidate answer set we mean a consistent set of ground regular literals satisfying the rules of the program.

$A_2 = \{p(1)\}$. In A_1 , $S = \emptyset$ which contradicts the rule. In A_2 , $S = \{1\}$ but this would contradict the $\mathcal{A}log$'s VCP. The program is inconsistent³.

We hope that the examples are sufficient to show how the informal semantics of $\mathcal{A}log$ can give a programmer some guidelines in avoiding formation of sets problematic from the standpoint of VCP. In what follows we

- Expand $\mathcal{A}log$ by allowing infinite sets and several additional set related constructs useful for knowledge representation and teaching.
- Propose an alternative formalization of the original VCP and incorporate it into the semantics of new language, $\mathcal{S}log^+$, which allows more liberal construction of sets and their use in programming rules. (The name of the new language is explained by its close relationship with language $\mathcal{S}log$ [34] – see Theorem 2).
- Show that, for programs without disjunction and infinite sets, the formal semantics of aggregates in $\mathcal{S}log^+$ coincides with that of several other known languages. Their intuitive and formal semantics, however, are based on quite different ideas and seem to be more involved than that of $\mathcal{S}log^+$.
- Prove some basic properties of programs in (extended) $\mathcal{A}log$ and $\mathcal{S}log^+$.

2 Syntax and Semantics of $\mathcal{A}log$

In what follows we retain the name $\mathcal{A}log$ for the new language and refer to the earlier version as “original $\mathcal{A}log$ ”.

2.1 Syntax

Let Σ be a (possibly sorted) signature with a finite collection of predicate and function symbols and (possibly infinite) collection of object constants, and let \mathcal{A} be a finite collection of symbols used to denote functions from sets of terms of Σ into integers. Terms and literals over signature Σ are defined as usual and referred to as *regular*. Regular terms are called *ground* if they contain no variables and no occurrences of symbols for arithmetic functions. Similarly for literals. We refer to an expression

$$\{\bar{X} : cond\} \tag{1}$$

where *cond* is a finite collection of regular literals and \bar{X} is the list of variables occurring in *cond*, as a *set name*. It is read as *the set of all objects of the program believed to satisfy cond*. Variables from \bar{X} are often referred to as *set variables*. An occurrence of a set variable in (1) is called *bound* within (1). Since treatment of variables in extended $\mathcal{A}log$ is the same as in the original language we limit our

³ There is a common argument for the semantics in which $\{p(1)\}$ would be the answer set of P_2 : “Since $card\{X : p(X)\} \geq 0$ is always true it can be dropped from the rule without changing the rule’s meaning”. But the argument assumes existence of the set denoted by $\{X : p(X)\}$ which is not always the case in $\mathcal{A}log$.

attention to programs in which every occurrence of a variable is bound. Rules containing non-bound occurrences of variables are considered as shorthands for their ground instantiations (for details see [15]).

A *set atom* of \mathcal{Alog} is an expression of the form

$$f_1(S_1) \odot f_2(S_2) \quad (2)$$

or

$$f(S) \odot k \quad (3)$$

where f, f_1, f_2 are functions from \mathcal{A} , S, S_1, S_2 are set names, k is a number, and \odot is an arithmetic relation $>, \geq, <, \leq, =$ or \neq , or of the form

$$S_1 \otimes S_2 \quad (4)$$

where \otimes is \subset, \subseteq , or $=$. We often write $f(\{\bar{X} : p(\bar{X})\})$ as $f\{\bar{X} : p(\bar{X})\}$ and $\{\bar{X} : p(\bar{X})\} \otimes S$ and $S \otimes \{\bar{X} : p(\bar{X})\}$ as $p \otimes S$ and $S \otimes p$ respectively. Regular and set atoms are referred to as *atoms*. A *rule* of \mathcal{Alog} is an expression of the form

$$head \leftarrow body \quad (5)$$

where *head* is a disjunction of regular literals or a set atom of the form $p \subseteq S$, $S \subseteq q$, or $p = S$, and *body* is a collection of regular literals (possibly preceded by *not*) and set atoms. A rule with set atom in the head is called *set introduction rule*. Note that both head and body of a rule can be infinite. All parts of \mathcal{Alog} rules, including *head*, can be empty. A *program* of \mathcal{Alog} is a collection of \mathcal{Alog} 's rules.

2.2 Semantics

To define the semantics of \mathcal{Alog} programs we first notice that the *standard definition of answer set from [13] is applicable to programs with infinite rules*. Hence we already have the definition of answer set for \mathcal{Alog} programs not containing occurrences of set atoms. We also need the satisfiability relation for set atoms. Let A be a set of ground regular literals. If $f(\{\bar{t} : cond(\bar{t}) \subseteq A\})$ is defined then $f(\{\bar{X} : cond\}) \geq k$ is satisfied by A (is *true* in A) iff $f(\{\bar{t} : cond(\bar{t}) \subseteq A\}) \geq k$. Otherwise, $f(\{\bar{X} : cond\}) \geq k$ is falsified (is *false* in A). If $f(\{\bar{t} : cond(\bar{t}) \subseteq A\})$ is not defined then $f(\{\bar{X} : cond\}) \geq k$ is *undefined* in A . (For instance, atom $card\{X : p(X)\} \geq 0$ is undefined in A if A contains an infinite collection of atoms formed by p .) Similarly for other set atoms. Finally a rule is *satisfied* by S if its head is *true* in S or its body is *false* or *undefined* in S .

Answer Sets for Programs without Set Introduction Rules. To simplify the presentation we first give the definition of answer sets for programs whose rules contain no set atoms in their heads. First we need the following definition:

Definition 1 (Set Reduct of Alog). Let Π be a ground program of *Alog*. The set reduct of Π with respect to a set of ground regular literals A is obtained from Π by

1. removing rules containing set atoms which are false or undefined in A .
2. replacing every remaining set atom SA by the union of $\text{cond}(\bar{t})$ such that $\{\bar{X} : \text{cond}(\bar{X})\}$ occurs in SA and $\text{cond}(\bar{t}) \subseteq A$.

The first clause of the definition removes rules useless because of the truth values of their aggregates in A . The next clause reflects the principle of avoiding vicious circles. Clearly, set reducts do not contain set atoms.

Definition 2 (Answer Set). A set A of ground regular literals over the signature of a ground *Alog* program Π is an answer set of Π if A is an answer set of the set reduct of Π with respect to A .

It is easy to see that for programs of the original *Alog* our definition coincides with the old one. Next several examples demonstrate the behavior of our semantics for programs not covered by the original syntax.

Infinite Universe

Example 3 (Aggregates on infinite sets). Consider a program E_1 consisting of the following rules:

```
even(0).
even(I+2) :- even(I).
q :- min{X : even(X)} = 0.
```

It is easy to see that the program has one answer set, $S_{E_1} = \{q, \text{even}(0), \text{even}(2), \dots\}$. Indeed, the reduct of E_1 with respect to S_{E_1} is the infinite collection of rules

```
even(0).
even(2) :- even(0).
...
q :- even(0), even(2), even(4) ...
```

The last rule has the infinite body constructed in the last step of definition 1. Clearly, S_{E_1} is a subset minimal collection of ground literals satisfying the rules of the reduct (i.e. its answer set). Hence S_{E_1} is an answer set of E_1 .

Example 4 (Programs with undefined aggregates). Now consider a program E_2 consisting of the rules:

```
even(0).
even(I+2) :- even(I).
q :- card{X : even(X)} > 0.
```

This program has one answer set, $S_{E_2} = \{\text{even}(0), \text{even}(2), \dots\}$. Since our aggregates range over natural numbers, the aggregate *card* is not defined on the set $\text{card}\{t : \text{even}(t) \in S_{E_2}\}$. This means that the body of the last rule is undefined. According to clause one of definition 1 this rule is removed. The reduct of E_2 with respect to S_{E_2} is

```

even(0).
even(2) :- even(0).
even(4) :- even(2).
...

```

Hence S_{E_2} is the answer set of E_2 .⁴ It is easy to check that, since every set A satisfying the rules of E_2 must contain all even numbers, S_{E_1} is the only answer set.

Programs with Set Atoms in the Bodies of Rules

Example 5 (Set atoms in the rule body). Consider a knowledge base containing two complete lists of atoms:

```

taken(mike,cs1).  taken(mike,cs2).  taken(john,cs2).
required(cs1).   required(cs2).

```

Set atoms allow for a natural definition of the new relation, *ready_to_graduate*(S), which holds if student S has taken all the required classes from the second list:

```

ready_to_graduate(S) :- {C: required(C)}  $\subseteq$  {C: taken(S,C)}.

```

The intuitive meaning of the rule is reasonably clear. The program consisting of this rule and the closed world assumption:

```

-ready_to_graduate(S) :- not ready_to_graduate(S)

```

implies that Mike is ready to graduate while John is not. If the list of classes taken by a student is incomplete the closed world assumption should be removed but the first rule still can be useful to determine people who are definitely ready to graduate. Even though the story can be represented in ASP without the set atoms, such representations are substantially less intuitive and less elaboration tolerant. Here is a simplified example of alternative representation suggested to the authors by a third party:

```

ready_to_graduate :- not -ready_to_graduate.
-ready_to_graduate :- not taken(c).

```

(Here student is eliminated from the parameters and we are limited to only one required class, c .) Even though in this case the answers are correct, unprincipled use of default negation leads to some potential difficulties. Suppose, for instance, that a student may graduate if given a special permission. This can be naturally added as a rule

```

ready_to_graduate :- permitted.

```

If the program is expanded by `permitted` it becomes inconsistent. This, of course, is unintended and contradicts our intuition. No such problem exists for the original representation.

⁴ Of course this is true only because of our (somewhat arbitrary) decision to limit aggregates of *Alog* to those ranging over natural numbers. We could, of course, allow aggregates mapping sets into ordinals. In this case the body of the last rule of E_2 will be defined and the only answer set of E_2 will be S_{E_1} .

The next example shows how the semantics deals with vicious circles.

Example 6 (Set atoms in the rule body). Consider a program P_4

$p(a) :- p \subseteq \{X : q(X)\}.$
 $q(a).$

in which definition of $p(a)$ depends on the existence of the set denoted by $\{X : p(X)\}$. In accordance with the vicious circle principle no answer set of this program can contain $p(a)$. There are only two candidates for answer sets of P_4 : $S_1 = \{q(a)\}$ and $S_2 = \{q(a), p(a)\}$. The set atom reduct of P_4 with respect to S_1 is

$p(a) :- q(a).$
 $q(a).$

while set atom reduct of P_4 with respect to S_2 is

$p(a) :- p(a), q(a).$
 $q(a).$

Clearly, neither S_1 nor S_2 is an answer set of P_4 . As expected, the program is inconsistent.

Programs with Set Introduction Rules. A set introduction rule with head $p \subseteq S$ (where p is a predicate symbol and S is a set name) defines set p as an arbitrary subset of S ; rule with head $p = S$ simply gives S a different name; $S \subseteq p$ defines p as an arbitrary superset of S .

Example 7 (Set introduction rule). According to this intuitive reading the program P_9 :

$q(a).$
 $p \subseteq \{X : q(X)\}.$

has answer sets $A_1 = \{q(a)\}$ where the set p is empty and $A_2 = \{q(a), p(a)\}$ where $p = \{a\}$.

The formal definition of answer sets of programs with set introduction rules is given via a notion of *set introduction reduct*. (The definition is similar to that presented in [10]).

Definition 3 (Set Introduction Reduct). *The set introduction reduct of a ground Alog program Π with respect to a set of ground regular literals A is obtained from Π by*

1. replacing every set introduction rule of Π whose head is not true in A by

$\leftarrow body.$

2. replacing every set introduction rule of Π whose head $p \subseteq \{\bar{X} : q(\bar{X})\}$ (or $p = \{\bar{X} : q(\bar{X})\}$ or $\{\bar{X} : q(\bar{X})\} \subseteq p$) is true in A by

$$p(\bar{t}) \leftarrow \text{body}$$

for each $p(\bar{t}) \in A$.

Set A is an answer set of Π if it is an answer set of the set introduction reduct of Π with respect to A .

Example 8 (Set introduction rule). Consider a program P_9 from Example 7. The reduct of this program with respect to $A_1 = \{q(a)\}$ is $\{q(a)\}$ and hence A_1 is an answer set of P_9 . The reduct of P_9 with respect to $A_2 = \{q(a), p(a)\}$ is $\{q(a), p(a)\}$ and hence A_2 is also an answer set of P_9 . There are no other answer sets.

The use of a set introduction rule $p \subseteq S \leftarrow \text{body}$ is very similar to that of choice rule $\{p(\bar{X}) : q(\bar{X})\} \leftarrow \text{body}$ of [24] implemented in Clingo and other similar systems. In fact, if p from the set introduction rule does not occur in the head of any other rule of the program, the two rules have the same meaning. However if this condition does not hold the meaning is different. An *Alg* program consisting of rules $p \subseteq \{X : q_1(X)\}$ and $p \subseteq \{X : q_2(X)\}$ defines an arbitrary set p from the intersection of q_1 and q_2 . With choice rules it is not the case. We prefer the set introduction rule because of its more intuitive reading (after all everyone is familiar with the statement “ p is an arbitrary subset of q ”) and relative simplicity of the definition of its formal semantics as compared with that of the choice rule.

Our last example shows how subset introduction rule with equality can be used to represent synonyms:

Example 9 (Synonyms). Suppose we have a set of cars represented by atoms formed by a predicate symbol *car*, e.g., $\{car(a), car(b)\}$. The following rule

`carro = {X:car(X)} :- spanish.`

allows to introduce a new name of this set for Spanish speaking people. Clearly, *car* and *carro* are synonyms. Hence, program $P_9 \cup \{\text{spanish}\}$ has one answer set: $\{\text{spanish}, car(a), car(b), carro(a), carro(b)\}$.

3 Alternative Formalization of VCP – Language *Slog*⁺

In this section we introduce alternative interpretation of VCP (referred to as *weak VCP*) and incorporate it in the semantics of a new logic programming language with set, called *Slog*⁺. The syntax of *Slog*⁺ coincides with that of *Alg*. Its informal semantics is based on weak VCP. By $C(T)$ we denote a set atom containing an occurrence of set term T . The *instantiation* of $C(\{X : p(X)\})$ in a set A of regular literals obtained from $C(\{X : p(X)\})$ by replacing $\{X : p(X)\}$ by $\{t : p(t) \in A\}$. The weak VCP is: *belief in $p(t)$ (i.e. inclusion of $p(t)$ in an answer set A) must be established without reference to the instantiation of a set atom C in A unless the truth of this instantiation can be demonstrated without reference to $p(t)$.*

Example 10. To better understand the weak VCP, let us consider program

```
p(0) :- C.
:- not p(0).
```

First we assume C be $\text{card}\{X : p(X)\} > 0$. There is only one candidate answer set $A = \{p(0)\}$ for this program. Belief in $p(0)$ (i.e. its membership in answer set A) can only be established by checking if instantiation $\text{card}\{t : p(t) \in A\} > 0$ of C in A holds. This is prohibited by weak VCP unless the truth of this instantiation can be demonstrated without reference to $p(0)$. But this cannot be so demonstrated because $\text{card}\{t : p(t) \in A\} > 0$ holds only when $p(0)$ is in A . Hence, A is not an answer set. Now let C be $\text{card}\{X : p(X)\} \geq 0$. This time the truth of instantiation $\text{card}\{t : p(t) \in A\} \geq 0$ of C can be demonstrated without reference to $p(0)$ – the instantiation would be true even if A were empty. Hence $p(0)$ must be believed and thus the program has one answer set, $\{p(0)\}$.

To make weak VCP based semantics precise we need the following notation and definitions: By \bar{W}^n, \bar{V}^n we denote n -ary vectors of sets of ground regular literals and by W_i, V_i their i -th coordinates. $\bar{W}^n \leq \bar{V}^n$ if for every i , $W_i \subseteq V_i$. $\bar{W}^n < \bar{V}^n$ if $\bar{W}^n \leq \bar{V}^n$ and $\bar{W}^n \neq \bar{V}^n$. A set atom $C(\{X : p_1(X)\}, \dots, \{X : p_n(X)\})$ is *satisfied* by \bar{W}^n if $C(\{t : p_1(t) \in W_1\}, \dots, \{t : p_n(t) \in W_n\})$ is true.

Definition 4 (Minimal Support). Let A be a set of ground regular literals of Π , and C be a set atom with n parameters. \bar{W}^n is a minimal support for C in A if

- For ever $1 \leq i \leq n$, $W_i \subseteq A$.
- Every \bar{V}^n such that for every $1 \leq i \leq n$, $W_i \subseteq V_i \subseteq A$ satisfies C .
- No $\bar{U}^n < \bar{W}^n$ satisfies the first two conditions.

Intuitively, the weak VCP says that set atom C can be safely used to support the reasoner's beliefs iff the existence of a minimal support of C can be established without reference to those beliefs. Precise definition of answer sets of $Slog^+$ is obtained by replacing definition 1 of set reduct of \mathcal{Alog} by definition 5 below and combining it with definition 3.

Definition 5 (Set-reduct of $Slog^+$). A set reduct of $Slog^+$ program Π with respect to a set A of ground regular literals is obtained from Π by

1. Removing rules containing set atoms which are false or undefined in A .
2. Replacing every remaining set atom C in the body of the rule by the union of coordinates of one of its minimal supports.

Clearly such a reduct is a regular ASP program without sets. A is an answer set of a $Slog^+$ program Π if A is an answer set of a weak set reduct of Π with respect to A .

Example 11. Consider now an $Slog^+$ program P_3

```

p(3) :- card{X : p(X)} >= 2.
p(2) :- card{X : p(X)} >= 2.
p(1).

```

It has two candidate answer sets: $A_1 = \{p(1)\}$ and $A_2 = \{p(1), p(2), p(3)\}$. In A_1 the corresponding condition is not satisfied and, hence, the weak set reduct of the program with respect to A_1 is $p(1)$. Consequently, A_1 is an answer set of P_3 . In A_2 the condition has three minimal supports: $M_1 = \{p(1), p(2)\}$, $M_2 = \{p(1), p(3)\}$, and $M_3 = \{p(2), p(3)\}$. Hence, the program has nine weak set reducts of P_3 with respect to A_2 . Each reduct is of the form

```

p(3) :- Mi.
p(2) :- Mj.
p(1).

```

where M_i and M_j are minimal supports of the condition. Clearly, the first two rules of such a reduct are useless and hence A_2 is not an answer set of this reduct. Consequently A_2 is not an answer set of P_3 .

The following two results help to better understand the semantics of $Slog^+$.

Theorem 1. *If a set A is an $Alog$ answer set of Π then A is an $Slog^+$ answer set of Π .*

As an $Slog^+$ program, P_2 has an answer set of $\{p(1)\}$, but it has no answer set as an $Alog$ program. The following result shows that there are many such programs and justifies our name for the new language.

Theorem 2. *Let Π be a program which, syntactically, belongs to both $Slog$ and $Slog^+$. A set A is an $Slog$ answer set of Π iff it is an $Slog^+$ answer set of Π .*

As shown in [35] $Slog$ has sufficient expressive power to formalize complex forms of recursion, including that used in the Company Control Problem [6]. Theorem 2 guarantees that the same representations will work in $Slog^+$. Of course, in many respects $Slog^+$ substantially increases the expressive power of $Slog$. Most importantly it expands the $Slog$ semantics to programs with epistemic disjunction – something which does not seem to be easy to do using the original definition of $Slog$ answer sets. Of course, new set constructs and rules with infinite number of literals are available in $Slog^+$ but not in $Slog$. On another hand, $Slog$ allows multisets – a feature we were not trying to include in our language. The usefulness of multisets and the analysis of its cost in terms of growing complexity of the language due to its introduction is still under investigation.

Unfortunately, the additional power of $Slog^+$ as compared with $Alog$ comes at a price. Part of it is a comparative complexity of the definition of $Slog^+$ set reduct. But, more importantly, the formalization of the weak VCP does not eliminate all the known paradoxes of reasoning with sets. Consider, for instance the following example:

Example 12. Recall program P_2 :

$p(1) :- \text{card}\{X:p(X)\} \geq 0.$

from Example 1 and assume, for simplicity, that parameters of p are restricted to $\{0, 1\}$. Viewed as a program of \mathcal{Alog} , P_2 is inconsistent. In \mathcal{Slog}^+ (and hence in \mathcal{Slog} and \mathcal{Flog} (the language defined in [6])) it has an answer set $\{p(1)\}$. The latter languages therefore admit existence of set $\{X : p(X)\}$. Now let us look at program P_5 :

$p(1) :- \text{card}\{X : p(X)\} = Y, Y \geq 0.$

and its grounding P_6 :

$p(1) :- \text{card}\{X:p(X)\} = 1, 1 \geq 0.$

$p(1) :- \text{card}\{X:p(X)\} = 0, 0 \geq 0.$

They seem to express the same thought as P_2 , and it is natural to expect all these programs to be equivalent. It is indeed true in \mathcal{Alog} – none of the programs is consistent. According to the semantics of \mathcal{Slog}^+ (and \mathcal{Slog} and \mathcal{Flog}), however, P_5 and P_6 are inconsistent. To see that notice that there are two candidate answer sets for P_6 : $A_1 = \emptyset$ and $A_2 = \{p(1)\}$. The minimal support of $\text{card}\{X : p(X)\} = 0$ in A_1 is \emptyset and hence the only weak set reduct of P_6 with respect to A_1 is $\{p(1) :- 0 \geq 0\}$. A_1 is not an answer set of P_6 . The minimal support of $\text{card}\{X : p(X)\} = 1$ in A_2 is $\{p(1)\}$. The only weak set reduct is $\{p(1) :- p(1), 1 \geq 0\}$. A_2 is not an answer set of P_6 either. It could be that this paradoxical behavior will be in the future explained from some basic principles but currently authors are not aware of such an explanation.

4 Properties of VCP Based Extensions of ASP

In this section we give some basic properties of \mathcal{Alog} and \mathcal{Slog}^+ programs. Propositions 1 and 2 ensure that, as in regular ASP, answer sets of \mathcal{Alog} program are formed using the program rules together with the rationality principle. Proposition 3 is the $\mathcal{Alog}/\mathcal{Slog}^+$ version of the Splitting Set Theorem – basic technical tool used in theoretical investigations of ASP and its extensions [14,19,38].

Proposition 1 (Rule Satisfaction and Supportedness). *Let A be an \mathcal{Alog} or \mathcal{Slog}^+ answer set of a ground program Π . Then*

- A satisfies every rule r of Π .
- If $p(\bar{t}) \in A$ then there is a rule r from Π such that the body of r is satisfied by A and
 - $p(\bar{t})$ is the only atom in the head of r which is true in A or
 - the head of r is of the form $p \odot \{\bar{X} : q(\bar{X})\}$ and $q(\bar{t}) \in A$. (It is often said that rule r supports atom p .)

By the intuitive and formal meaning of set introduction rules, the anti-chain property no longer holds. However, the anti-chain property still holds for programs without set atoms in the heads of their rules.

Proposition 2 (Anti-chain Property). *If Π is a program without set atoms in the heads of its rules then there are no Alog answer sets A_1, A_2 of Π such that $A_1 \subset A_2$. Similarly for its $Slog^+$ answer sets.*

Before formulating the next result we need some terminology.

Definition 6 (Occurrences of Regular Literals in Aggregate Atoms).

We say that a ground literal l occurs in a set atom C if there is a set name $\{X : \text{cond}(X)\}$ occurring in C and l is a ground instance of some literal in cond . If B is a set of ground literals possibly preceded by default negation not then l occurs in B if $l \in B$, or not $l \in B$, or l occurs in some set atom from B .

Definition 7 (Splitting Set). *Let Π be a program with signature Σ . A set S of ground regular literals of Σ is called a splitting set of Π if, for every rule r of Π , if l occurs in the head of r then every literal occurring in the body of r belongs to S . The set of rules of Π constructed from literals of S is called the bottom of Π relative to S ; the remaining rules are referred to as the top of Π relative to S .*

Note that the definition implies that no literal occurring in the bottom of Π relative to S can occur in the heads of rules from the top of Π relative to S .

Proposition 3 (Splitting Set Theorem). *Let Π be a ground program, S be its splitting set, and Π_1 and Π_2 be the bottom and the top of Π relative to S respectively. Then a set A is an answer set of Π iff $A \cap S$ is an answer set of Π_1 and A is an answer set of $(A \cap S) \cup \Pi_2$.*

Note that this formulation differs from the original one in two respects. First, rules of the program can be infinite. Second, the definition of occurrence of a regular literal in a rule changes to accommodate the presence of set atoms.

5 Related Work

There are multiple approaches to introducing aggregates in logic programming languages under the answer sets semantics [17,10,24,23,22,26,27,28,7,8,25,34,18,33,21,6,30,20,39,16,32,15,9,2]. In addition to this work our paper was significantly influenced by the original work on VCP in set theory and principles of language design advocated by Dijkstra, Hoare, Wirth and others. Harrison et al's work [16] explaining the semantics of some constructs of gringo in terms of infinitary formulas of Truszczyński [37] led to their inclusion in *Alog* and *Slog*⁺. The notion of set reduct of *Alog* was influenced by the reduct introduced for defining the semantics of Epistemic Specification in [11]. Recent work by Alviano and Faber [1] helped us to realize the close relationship between *Alog* and *Slog* and Argumentation theory [5,3,36] which certainly deserves further investigation, as well as provided us with additional knowledge about *Alog*. More information about *Slog* and *Slog*⁺ can be found in Section 3. Shen et al. [33] and Liu et al. [20] propose equivalent semantics for disjunctive constraint programs (i.e., programs with rules whose bodies

are built from constraint atoms and whose heads are epistemic disjunctions of such atoms). This generalizes the standard ASP semantics for disjunctive programs. We conjecture that when we adapt our definition of $Slog^+$ semantics to disjunctive constraint programs, it will coincide with that of [33,20]. However, our definition seems to be simpler and is based on clear, VCP related intuition.

6 Conclusion

The paper belongs to the series of works aimed at the development of an answer set based knowledge representation language. Even though we want to have a language suitable for serious applications our main emphasis is on teaching. This puts additional premium on clarity and simplicity of the language design. In particular we believe that the constructs of the language should have a simple syntax and a clear intuitive semantics based on understandable informal principles. In our earlier paper [15] we concentrated on a language $Alog$ expanding standard Answer Set Prolog by aggregates. We argued that the syntax of the language is simpler than that of the most popular aggregate language $Flog$ implemented in Clingo and other similar systems. In particular, $Alog$'s notion of grounding *allows to define the intuitive (and formal) meaning of a set name independently from its occurrence in a rule*. As the result, set name $\{X : p(X)\}$ can be always equivalently replaced by $\{Y : p(Y)\}$. In $Flog$, it is not the case. A semantics of aggregates in $Alog$ was based on a particularly simple and restrictive formalization of VCP. In this paper we:

- Expanded syntax and semantics of the original $Alog$ by allowing
 - rules with an infinite number of literals – a feature of theoretical interest also useful for defining aggregates on infinite sets;
 - subset relation between sets in the bodies of rules concisely expressing a specific form of universal quantification;
 - set introduction – a feature with functionality somewhat similar to that of the choice rule of clingo but with different intuitive semantics.

Our additional set constructs are aimed at showing that our original languages can be expanded in a natural and technically simple ways. Other constructs such as set operations and rules with variables ranging over sets (in the style of [4]), etc. are not discussed. Partly this is due to space limitations – we do not want to introduce any new constructs without convincing examples of their use. The future will show if such extensions are justified.

- Introduced a new KR language, $Slog^+$, with the same syntax as $Alog$ but different semantics for the set related constructs. The new language is less restrictive and allows formation of substantially larger collection of sets. Its semantics is based on the alternative, weaker formalization of VCP.
- Proved that (with the exception of multisets) $Slog^+$ is an extension of a well known aggregate language $Slog$. The semantics of the new language is based on the intuitive idea quite different from that of $Slog$ and the definition of its semantics is simpler. We point out some paradoxes of $Slog^+$ (and $Flog$) which prevent us from advocating them as standard ASP language with aggregates.

- Proved a number of basic properties of programs of \mathcal{Alog} and \mathcal{Slog}^+ .

References

1. Alviano, M., Faber, W.: Stable model semantics of abstract dialectical frameworks revisited: A logic programming perspective. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence. IJCAI Organization, Buenos Aires, Argentina. pp. 2684–2690 (2015)
2. Alviano, M., Leone, N.: Complexity and compilation of gz-aggregates in answer set programming. *Theory and Practice of Logic Programming* 15(4-5), 574–587 (2015)
3. Brewka, G., Ellmauthaler, S., Strass, H., Wallner, J.P., Woltran, S.: Abstract dialectical frameworks revisited. In: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence. pp. 803–809. AAAI Press (2013)
4. Dovier, A., Pontelli, E., Rossi, G.: Intensional sets in CLP. In: Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings. pp. 284–299 (2003)
5. Dung, P.M.: On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. *Artificial intelligence* 77(2), 321–357 (1995)
6. Faber, W., Pfeifer, G., Leone, N.: Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence* 175(1), 278–298 (2011)
7. Ferraris, P.: Answer sets for propositional theories. In: LPNMR. pp. 119–131 (2005)
8. Ferraris, P., Lifschitz, V.: Weight constraints as nested expressions. *TPLP* 5(1-2), 45–74 (2005)
9. Gebser, M., Harrison, A., Kaminski, R., Lifschitz, V., Schaub, T.: Abstract gringo. *Theory and Practice of Logic Programming* 15(4-5), 449–463 (2015)
10. Gelfond, M.: Representing Knowledge in A-Prolog. In: Kakas, A.C., Sadri, F. (eds.) *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*. vol. 2408, pp. 413–451. Springer Verlag, Berlin (2002)
11. Gelfond, M.: New semantics for epistemic specifications. In: *Logic Programming and Nonmonotonic Reasoning*, pp. 260–265. Springer (2011)
12. Gelfond, M., Kahl, Y.: *Knowledge Representation, Reasoning, and the Design of Intelligent Agents*. Cambridge University Press (2014)
13. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3/4), 365–386 (1991)
14. Gelfond, M., Przymusinska, H.: On consistency and completeness of autoepistemic theories. *Fundam. Inf.* 16(1) (Jan 1992)
15. Gelfond, M., Zhang, Y.: Vicious circle principle and logic programs with aggregates. *TPLP* 14(4-5), 587–601 (2014), <http://dx.doi.org/10.1017/S1471068414000222>
16. Harrison, A.J., Lifschitz, V., Yang, F.: The semantics of gringo and infinitary propositional formulas. In: *KR* (2014)
17. Kemp, D.B., Stuckey, P.J.: Semantics of logic programs with aggregates. In: *ISLP*. vol. 91, pp. 387–401. Citeseer (1991)
18. Lee, J., Lifschitz, V., Palla, R.: A reductive semantics for counting and choice in answer set programming. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008. pp. 472–479 (2008), <http://www.aaai.org/Library/AAAI/2008/aaai08-075.php>
19. Lifschitz, V., Turner, H.: Splitting a logic program. In: Proceedings of the 11th International Conference on Logic Programming (ICLP94). pp. 23–38 (1994)

20. Liu, G., Goebel, R., Janhunen, T., Niemelä, I., You, J.H.: Strong equivalence of logic programs with abstract constraint atoms. In: *Logic Programming and Non-monotonic Reasoning*, pp. 161–173. Springer (2011)
21. Liu, L., Pontelli, E., Son, T.C., Truszczyński, M.: Logic programs with abstract constraint atoms: The role of computations. *Artif. Intell.* 174(3-4), 295–315 (2010)
22. Marek, V.W., Remmel, J.B.: Set constraints in logic programming. In: *Logic Programming and Nonmonotonic Reasoning*, pp. 167–179. Springer (2004)
23. Marek, V.W., Truszczyński, M.: Logic programs with abstract constraint atoms. In: *AAAI*, vol. 4, pp. 86–91 (2004)
24. Niemela, I., Simons, P., Soinen, T.: Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2), 181–234 (Jun 2002)
25. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming* 7, 355–375 (2007)
26. Pelov, N.: Semantics of logic programs with aggregates. Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium (Apr 2004)
27. Pelov, N., Denecker, M., Bruynooghe, M.: Partial stable models for logic programs with aggregates. In: *LPNMR*, pp. 207–219 (2004)
28. Pelov, N., Truszczyński, M.: Semantics of disjunctive programs with monotone aggregates - an operator-based approach. In: *NMR*, pp. 327–334 (2004)
29. Poincare, H.: *Les mathematiques et la logique*. *Review de metaphysique et de morale* 14, 294–317 (1906)
30. Pontelli, E., Son, T.C., Tu, P.H.: Answer sets for logic programs with arbitrary abstract constraint atoms. *CoRR abs/1110.2205* (2011)
31. Russell, B.: Mathematical logic as based on the theory of types. *American Journal of Mathematics* 30(3), 222–262 (1908)
32. Shen, Y.D., Wang, K., Eiter, T., Fink, M., Redl, C., Krennwallner, T., Deng, J.: Flp answer set semantics without circular justifications for general logic programs. *Artificial Intelligence* 213, 1–41 (2014)
33. Shen, Y., You, J., Yuan, L.: Characterizations of stable model semantics for logic programs with arbitrary constraint atoms. *TPLP* 9(4), 529–564 (2009)
34. Son, T.C., Pontelli, E.: A constructive semantic characterization of aggregates in answer set programming. *TPLP* 7(3), 355–375 (2007)
35. Son, T.C., Pontelli, E., Tu, P.H.: Answer sets for logic programs with arbitrary abstract constraint atoms. *J. Artif. Intell. Res. (JAIR)* 29, 353–389 (2007)
36. Strass, H.: Approximating operators and semantics for abstract dialectical frameworks. *Artificial Intelligence* 205, 39–70 (2013)
37. Truszczyński, M.: Connecting first-order asp and the logic fo (id) through reducts. In: *Correct Reasoning*, pp. 543–559. Springer (2012)
38. Turner, H.: Splitting a default theory. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1*, pp. 645–651 (1996)
39. Wang, Y., Lin, F., Zhang, M., You, J.H.: A well-founded semantics for basic logic programs with arbitrary abstract constraint atoms. In: *AAAI* (2012)